**U.S. Non-Provisional Patent Application**

**Attorney Docket No.: 200314673-1**

**Title:**

# CLUSTER-BASED DISK BACKUP AND RESTORATION

**Inventors:**

**Gunnar Paul Seaburg**
11 Moonseed Place
The Woodlands, TX 77381
Citizenship: USA

**Tri Minh Nguyen**
16506 Westwego Trail
Cypress, TX 77429
Citizenship: USA

**Christoph J. Graham**
11407 Parkriver
Houston, TX 77070
Citizenship: USA

# CLUSTER-BASED DISK BACKUP AND RESTORATION

## BACKGROUND

**[0001]** Sometimes computer users may want to have their operating system, file system, applications, and/or data re-installed, or reset to their original factory settings. Additionally and/or alternatively, a computer user may want to backup and/or restore a partition on a hard disk drive. To perform a restore, conventionally, a user may install a "boot disk" to boot the target computer and then reinstall from a set of disks. The operating system, applications, and so on may be reinstalled on a file by file basis using file system programs, data structures, and so on. These file system processes, data structures and so on, are typically designed for file system specific special purposes and thus may examine directory trees, manipulate file allocation tables (FATs), check file protections, search for free space, attempt to optimize directory structures, and so on, all of which take processing cycles and thus time.

**[0002]** Being able to restore from a set of disks presupposes that a set of disks from which the restore can proceed exists. Thus, conventional restore systems are typically associated with a backup system for creating the set of disks from which the reinstall can occur. The backup system may be employed "at the factory" (e.g., before a system is shipped) and/or later (e.g., on a user driven timetable). Like the restore systems and methods, the backup systems and methods may use file system programs, data structures, and so on, and thus may experience the same processing time issues.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0003]** The accompanying drawings, which are incorporated in and constitute a part of the specification, illustrate various example systems, methods, and so on that illustrate various example embodiments of aspects of the invention. It will be appreciated that the illustrated element boundaries (e.g., boxes, groups of boxes, or other shapes) in the figures represent one example of the boundaries. One of ordinary skill in the art will appreciate that one element may be designed as multiple elements or that multiple elements may be designed as one element. An element shown as an internal component of another element may be implemented as an external component and vice versa. Furthermore, elements may not be drawn to scale.

[0004] Figure 1 illustrates an example cluster-based backup method.

[0005] Figure 2 illustrates another example cluster-based backup method.

[0006] Figure 3 illustrates an example cluster-based restoration method.

[0007] Figure 4 illustrates another example cluster-based restoration method.

[0008] Figure 5 illustrates an example system for performing cluster-based backups and/or restores.

[0009] Figure 6 illustrates an example computing environment in which example systems and methods illustrated herein can operate.

[0010] Figure 7 illustrates an example image forming device in which example systems and methods illustrated herein can operate.

[0011] Figure 8 illustrates an example application programming interface (API).

## DETAILED DESCRIPTION

[0012] This application describes systems, methods, and so on associated with backing up and/or restoring a hard drive partition. The example systems and methods perform cluster-based reads and/or writes of a partition rather than performing conventional file by file, file system processes on conventional data structures like files and so on. The example systems and methods employ a "backup partition" on a hard drive rather than external media for storing the backup/restore image. A cluster-based backup image may be placed on the hard drive "at the factory" before a computer is shipped. The cluster-based backup image may be placed, for example, in a separate partition from the partition being backed up on the hard drive and/or in a file in the partition being backed up on a hard drive. A cluster-based restore program may restore the user partition to its original state by doing cluster-based reads from the backup image and cluster-based writes to the partition to be restored, again without employing some conventional file system processes and data structures (e.g., files, directories).

[0013] Hard disk drives in computer systems (e.g., personal computers (PCs), servers) typically store data (bytes) in sectors (e.g., 512 bytes). A file system may then track which

sectors are used by files. For a large drive (e.g., 40 GB), a file system may therefore track a large number of sectors (e.g., 83,886,080 sectors). Thus, sectors may be grouped into "clusters" (e.g., groups of contiguous sectors) and the file system may track these clusters rather than sectors. In the FAT32 file system, a cluster size may be 16Kb (32 sectors). In the NTFS, a cluster size may default to 4Kb (8 sectors). Tracking by clusters facilitates reducing file system overhead but wastes space because a file, no matter how small, is likely to be allocated an entire cluster.

**[0014]** Regardless of the tradeoff between file system overhead and wasted space, data structures are populated with information about allocated clusters. For example, a volume bitmap may record which clusters contain data and which clusters are unallocated. Thus, the example systems and methods described herein may access the volume bitmap to determine which clusters to back up, and then use Windows® XP Pre-Install Environment (WinPE) processes to do cluster level reads and writes. In one example, the volume bitmap includes one bit per cluster on a hard drive. If the bit is set (e.g., 1), then the cluster is allocated and contains data to be backed up. If the bit is clear (e.g., 0), then the cluster is unallocated and does not need to be backed up. In one example, the volume bitmap can itself be stored in the backup image, to facilitate recreating a substantially exact image during a restore process. In the NTFS, the volume bitmap may include information about NTFS structures. But in the FAT32 system, the volume bitmap may not track FATs, and thus FAT data may need to be discovered and backed up separately.

**[0015]** The following includes definitions of selected terms employed herein. The definitions include various examples and/or forms of components that fall within the scope of a term and that may be used for implementation. The examples are not intended to be limiting. Both singular and plural forms of terms may be within the definitions.

**[0016]** As used in this application, the term "computer component" refers to a computer-related entity, either hardware, firmware, software, a combination thereof, or software in execution. For example, a computer component can be, but is not limited to being, a process running on a processor, a processor, an object, an executable, a thread of execution, a program, and a computer. By way of illustration, both an application running on a server and the server can be computer components. One or more computer components can reside within a process and/or thread of execution and a computer component can be localized on one computer and/or distributed between two or more computers.

[0017] "Computer-readable medium", as used herein, refers to a medium that participates in directly or indirectly providing signals, instructions and/or data. A computer-readable medium may take forms, including, but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media may include, for example, optical or magnetic disks and so on. Volatile media may include, for example, optical or magnetic disks, dynamic memory and the like. Transmission media may include coaxial cables, copper wire, fiber optic cables, and the like. Transmission media can also take the form of electromagnetic radiation, like that generated during radio-wave and infra-red data communications, or take the form of one or more groups of signals. Common forms of a computer-readable medium include, but are not limited to, a floppy disk, a flexible disk, a hard disk, a magnetic tape, other magnetic medium, a CD-ROM, other optical medium, punch cards, paper tape, other physical medium with patterns of holes, a RAM, a ROM, an EPROM, a FLASH-EPROM, or other memory chip or card, a memory stick, a carrier wave/pulse, and other media from which a computer, a processor or other electronic device can read. Signals used to propagate instructions or other software over a network, like the Internet, can be considered a "computer-readable medium."

[0018] "Logic", as used herein, includes but is not limited to hardware, firmware, software and/or combinations of each to perform a function(s) or an action(s), and/or to cause a function or action from another logic, method, and/or system. For example, based on a desired application or needs, logic may include a software controlled microprocessor, discrete logic like an application specific integrated circuit (ASIC), a programmed logic device, a memory device containing instructions, or the like. Logic may include one or more gates, combinations of gates, or other circuit components. Logic may also be fully embodied as software. Where multiple logical logics are described, it may be possible to incorporate the multiple logical logics into one physical logic. Similarly, where a single logical logic is described, it may be possible to distribute that single logical logic between multiple physical logics.

[0019] An "operable connection", or a connection by which entities are "operably connected", is one in which signals, physical communication flow, and/or logical communication flow may be sent and/or received. Typically, an operable connection includes a physical interface, an electrical interface, and/or a data interface, but it is to be noted that an operable connection may include differing combinations of these or other types

of connections sufficient to allow operable control. For example, two entities can be operably connected by being able to communicate signals to each other directly or through one or more intermediate entities like a processor, operating system, a logic, software, or other entity. Logical and/or physical communication channels can be used to create an operable connection.

**[0020]** "Signal", as used herein, includes but is not limited to one or more electrical or optical signals, analog or digital signals, data, one or more computer or processor instructions, messages, a bit or bit stream, or other means that can be received, transmitted and/or detected.

**[0021]** "Software", as used herein, includes but is not limited to, one or more computer or processor instructions that can be read, interpreted, compiled, and/or executed and that cause a computer, processor, or other electronic device to perform functions, actions and/or behave in a desired manner. The instructions may be embodied in various forms like routines, algorithms, modules, methods, threads, and/or programs including separate applications or code from dynamically linked libraries. Software may also be implemented in a variety of executable and/or loadable forms including, but not limited to, a stand-alone program, a function call (local and/or remote), a servelet, an applet, instructions stored in a memory, part of an operating system or other types of executable instructions. It will be appreciated by one of ordinary skill in the art that the form of software may be dependent on, for example, requirements of a desired application, the environment in which it runs, and/or the desires of a designer/programmer or the like. It will also be appreciated that computer-readable and/or executable instructions can be located in one logic and/or distributed between two or more communicating, co-operating, and/or parallel processing logics and thus can be loaded and/or executed in serial, parallel, massively parallel and other manners.

**[0022]** Suitable software for implementing the various components of the example systems and methods described herein include programming languages and tools like Java, Pascal, C#, C++, C, CGI, Perl, SQL, APIs, SDKs, assembly, firmware, microcode, and/or other languages and tools. Software, whether an entire system or a component of a system, may be embodied as an article of manufacture and maintained or provided as part of a computer-readable medium as defined previously. Another form of the software may include signals that transmit program code of the software to a recipient over a network or other communication medium. Thus, in one example, a computer-readable medium has a form of

signals that represent the software/firmware as it is downloaded from a web server to a user. In another example, the computer-readable medium has a form of the software/firmware as it is maintained on the web server. Other forms may also be used.

**[0023]** Some portions of the detailed descriptions that follow are presented in terms of algorithms and symbolic representations of operations on data bits within a memory. These algorithmic descriptions and representations are the means used by those skilled in the art to convey the substance of their work to others. An algorithm is here, and generally, conceived to be a sequence of operations that produce a result. The operations may include physical manipulations of physical quantities. Usually, though not necessarily, the physical quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated in a logic and the like.

**[0024]** It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like. It should be borne in mind, however, that these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise, it is appreciated that throughout the description, terms like processing, computing, calculating, determining, displaying, or the like, refer to actions and processes of a computer system, logic, processor, or similar electronic device that manipulates and transforms data represented as physical (electronic) quantities.

**[0025]** Example methods may be better appreciated with reference to the flow diagrams of **Figures 1** through **4**. While for purposes of simplicity of explanation, the illustrated methodologies are shown and described as a series of blocks, it is to be appreciated that the methodologies are not limited by the order of the blocks, as some blocks can occur in different orders and/or concurrently with other blocks from that shown and described. Moreover, less than all the illustrated blocks may be required to implement an example methodology. Furthermore, additional and/or alternative methodologies can employ additional, not illustrated blocks.

**[0026]** In the flow diagrams, blocks denote "processing blocks" that may be implemented with logic. A flow diagram does not depict syntax for any particular programming language, methodology, or style (e.g., procedural, object-oriented). Rather, a flow diagram illustrates

functional information one skilled in the art may employ to develop logic to perform the illustrated processing. It will be appreciated that in some examples, program elements like temporary variables, routine loops, and so on are not shown. It will be further appreciated that electronic and software applications may involve dynamic and flexible processes so that the illustrated blocks can be performed in other sequences that are different from those shown and/or that blocks may be combined or separated into multiple components. It will be appreciated that the processes may be implemented using various programming approaches like machine language, procedural, object oriented and/or artificial intelligence techniques.

[0027] **Figure 1** illustrates an example computerized cluster-based backup method **100**. The method **100** may include, at **110**, selecting a source partition (e.g., partition to be backed up) on a hard disk drive. A cluster-based backup image of the source partition will be produced and stored by the method **100** in a target partition on the hard disk drive. Thus, both the source partition (the partition to be backed up), and the target partition (the partition in which the cluster-based backup image will be written) are located on the same hard disk drive. In one example, the target partition may be located in the source partition, while in another example, the target partition is not located in the source partition. Thus, a cluster-based backup image may be stored in an otherwise inaccessible space on a user hard drive. The otherwise inaccessible space may be, for example, on a D drive (e.g., a second partition on a hard drive with a C drive) and/or on a "ghost" file located in the C drive. "Ghost file" and "ghost partition", as used herein, refer to entities similar to those described in U.S. Patent 5,974,567. For example, a ghost file may be a regular file that resides in a ghost partition. In one example, a ghost partition may be wrapped into a file so that a user and/or file system will not inadvertently overwrite the clusters associated with the ghost file. Furthermore, if a ghost partition is wrapped in file, then the file may be made non-relocatable by making it, for example, a read-only hidden system file. Wrapping a ghost partition in a file and making the file non-relocatable facilitate resolving issues associated with unintended overwriting of ghost partitions and/or files. A restore system or method can use knowledge of the location of the data clusters in the backup image to facilitate bypassing some file system specific calls, which can reduce the time required to do a restore when compared to a file by file based approach. In one example, WinPE functionality facilitates gathering cluster information and performing cluster-based acts like identifying a used cluster, reading a cluster, writing a cluster, and so on. These "lower level" (e.g., cluster based vs. file system based) WinPE

processes facilitate not using "higher level" Windows® processes that may be more file system oriented and thus slower. In one example, the cluster-based backup image may be created at a rate of more than five hundred megabytes per minute. While five hundred megabytes per minute are described, it is to be appreciated that other speeds can be employed.

**[0028]** The method **100** may also include, at **120**, obtaining a boot record for the source partition. In one example, the boot record may be obtained by reading the first sector from the source partition. The method **100** may also include, at **130**, examining the boot record to determine values for file system parameters (e.g., a file system type, a file system size, and a cluster size) for a file system stored in the source partition. Some example systems and methods may employ the WinPE environment and support Windows NT file system (NTFS) and/or FAT32 file systems while avoiding general purpose Windows XP file system calls. Thus, the file system may be, for example, an NTFS file system and/or a FAT32 file system.

**[0029]** The WinPE environment may also provide other functionalities. For example, the WinPE environment may provide APIs to processes that facilitate discovering disk partition layout, determining partition areas that contain data (e.g., used clusters) by querying a partition volume bitmap structure, and so on. These WinPE APIs may facilitate using WinPE processes to manipulate disk clusters in sequential on-disk order, from the beginning of the drive to the end, which may facilitate avoiding hard disk seek and latency times associated with file system based per file reading and writing. Additionally, backup and restore methods may use memory caching mechanisms to minimize seeks between user and backup areas.

**[0030]** The method **100** may also include, at **140**, calculating a value(s) for a source partition metadata parameter(s) from the file system parameters and/or source partition parameters. The source partition metadata parameters may store data concerning, for example, the source partition size, location, name, type, cluster-size, and so on.

**[0031]** The method **100** may also include, at **150**, writing a cluster-based backup image header data structure to the target partition. The cluster-based backup image header data structure may store data including, but not limited to, source partition metadata parameters, file system parameters, and source partition parameters. In one example, the cluster-based backup image header data structure stores a header length field, a format version field, an image backup file name field, a partition cluster count field, a backup image cluster count

field, a volume bitmap bit count field, a cluster size field, a sector size field, and a partition type field. While nine fields are described, it is to be appreciated that cluster-based backup image header data structures with a greater and/or lesser number of fields may be employed.

**[0032]** Not every file in a partition for which a cluster-based backup image will be produced may be included in the cluster-based backup image. For example, if the partition includes a cluster-based backup image, then it may not be desired to make a backup image of the backup image. Thus, the method **100** may include, at **160**, selectively manipulating bits in a volume bitmap associated with the source partition. The bits may be associated with files that are not to be included in the cluster-based backup image. For example, the bits may be associated with files like a hiberfil.sys file, a pagefile.sys file, and a ghost file.

**[0033]** The method **100** may also include, at **170**, identifying clusters in the source partition to be included in the cluster-based backup image. The clusters may be identified, for example, by accessing the volume bitmap. In one example, the volume bitmap may be accessed using functionality provided through a call like DeviceIoControl(..., FSCTL_GET_VOLUME_BITMAP, ...). At **180**, a determination is made concerning whether a particular cluster is to be backed up. If the determination at **180** is Yes, then at **182** a cluster is read from the source partition and, at **184**, is written to the cluster-based backup image. At **190**, a determination is made concerning whether there are more clusters to be considered for writing to the cluster-based backup image. If the determination is Yes, then processing returns to **180**, otherwise processing may conclude.

**[0034]** Various actions undertaken by method **100** may be accomplished using functionality provided by the WinPE environment. Thus, in one example, a boot record may be obtained and/or examined using a WinPE process. Similarly, the cluster-based backup image header data structure may be written to the target partition using a WinPE process and the volume bitmap may be accessed using a WinPE process. Likewise, a cluster may be read from the source partition using a WinPE process and a cluster may be written to the cluster-based backup image using a WinPE process. While WinPE processes are described, it is to be appreciated that other environments that provide cluster-based reads, writes, and so on may be employed.

**[0035]** While **Figure 1** illustrates various actions occurring in serial, it is to be appreciated that various actions illustrated in **Figure 1** could occur substantially in parallel.

By way of illustration, a first process could select a partition to be backed up and gather information about the partition (e.g., its metadata). Similarly, a second process could create the backup image header(s) and/or metadata data structures, while a third process could manipulate the volume bitmap, identify clusters, and control reading and writing the clusters to the cluster-based backup image. While three processes are described, it is to be appreciated that a greater and/or lesser number of processes could be employed and that lightweight processes, regular processes, threads, and other approaches could be employed.

[0036] **Figure 2** illustrates another example computerized method **200** for producing a cluster-based backup image of a source partition on a hard disk drive. The method **200** may include, at **210**, retrieving source partition parameters from a boot data structure associated with the source partition. The source partition parameters may include, for example, a file system type, a file system size, and a cluster size. The boot data structure may be, for example, a boot record. While three source partition parameters are described, it is to be appreciated that a greater and/or lesser number of source partition parameters may be acquired and/or written.

[0037] The method **200** may also include, at **220**, accessing a volume bitmap associated with the source partition to facilitate identifying clusters to be included in the cluster-based backup image. The volume bitmap may be accessed, for example, using WinPE functionality. The WinPE functionality may be invoked, for example, by using processes available through a WinPE API. The method **200** may also include, at **230**, writing to a target partition a cluster-based backup image metadata data structure configured to store partition metadata. Thus, the cluster-based backup image may contain clusters read from the source partition and data about those clusters and the image being created.

[0038] The method **200** may also include, at **250** through **270**, producing the cluster-based backup image. In one example, at **240**, a determination is made concerning whether there is another cluster to add to the cluster-based backup image. If the determination at **240** is No, then processing can conclude. But if the determination is Yes, then at **250** the method **200** may include selectively reading a cluster based, at least in part, on information stored in the volume bitmap. For example, if a volume bitmap bit is set (e.g., 1), then the cluster may be read but if the volume bitmap bit is clear (e.g., 0), then the cluster may not be read. After a cluster is read, the method **200** includes, at **260**, writing the cluster to the target partition. But a cluster-based backup image may not be simply a set of clusters in a partition.

Information about the clusters may be stored in a data structure(s) associated with the set of clusters. Therefore method **200** may include, at **270**, linking a cluster written to the target partition to the cluster-based backup image metadata data structure to facilitate restoring the source partition from the target partition. The linking may be accomplished by, for example, writing a cluster source identifier, a cluster destination identifier, a cluster number, and so on to the metadata data structure. The metadata data structure may also be updated with, for example, a count of the number of clusters written to the cluster-based backup image.

**[0039]** Like method **100** may employ functionality provided by the WinPE environment, so too may method **200**. Thus, in one example, the volume bitmap may be accessed using a WinPE process and clusters may be read and/or written using WinPE processes. In one example, the target partition to which the cluster-based backup image is written is in the source partition while in another example the target partition is not in the source partition.

**[0040]** While **Figure 2** illustrates various actions occurring in serial, it is to be appreciated that various actions illustrated in **Figure 2** could occur substantially in parallel. By way of illustration, a first process could retrieve partition parameters, access the volume bitmap and store a metadata data structure in the target partition. Similarly, a second process could read clusters from the source partition, a third process could write clusters to the target partition and a fourth process could link written clusters to the metadata data structure. While four processes are described, it is to be appreciated that a greater and/or lesser number of processes could be employed and that lightweight processes, regular processes, threads, and other approaches could be employed.

**[0041]** **Figure 3** illustrates an example computerized cluster-based partition restoration method **300**. The method **300** may include, at **310**, identifying a partition to restore on a hard disk drive. The partition may be a partition that a user desires to have restored to its "original factory settings". For example, a user may have tried to update an operating system registry and in so doing corrupted one or more files, links, data structures, applications and so on in a partition. The user may not be able to undo their actions and thus may want to start over with the computer configured the way it was the day it arrived. Conventionally, a partition may be restored from a set of compact disks and/or floppy drives. However, such a conventional technique may take hours and many user interactions. But if there is a cluster-based backup image available on the hard drive housing the partition to be restored, a cluster-based backup restoration method may be employed. This may facilitate, for example, performing the

restoration more quickly. In one example, the restoration may be accomplished in less than five minutes and require fewer user interactions.

**[0042]** Thus, the method **300** may include, at **320**, locating a cluster-based backup image(s) from which the partition can be restored. The cluster-based backup image(s) from which the partition can be restored is located on the hard disk drive having the partition to be restored. The method **300** may also include, at **330**, selecting a cluster-based backup image from which the partition will be restored. In some examples a restoration method may be configured to look for a single cluster-based backup image in a known location and thus actions performed at **320** and **330** may reflect the search for the single cluster-based backup image.

**[0043]** The method **300** may also include, at **340**, reading a cluster-based backup image header data structure from the cluster-based backup image. The header data structure may store, for example, partition parameters, file system parameters, and the like. In one example, partition parameters may include, but are not limited to, a partition type, a partition size, a partition location, a partition name, and so on. In another example, file system parameters include, but are not limited to, a file system type, a file system size, a cluster size, and so on.

**[0044]** A hard disk drive partition may be associated with a boot record. Thus, the method **300** may include, at **350**, writing partition parameters to a boot record associated with the partition and at **360** writing file system parameters to the boot record associated with the partition. Similarly, information about a partition (e.g., clusters in use) may be stored in a volume bitmap. Therefore, method **300** may also include, at **370**, resetting bits on a volume bitmap associated with the partition being restored. Thus, it is to be appreciated that a cluster-based backup image restoration process may include not only copying clusters of data, but also creating and/or populating data structures like a boot record and/or volume bitmap associated with the partition being restored.

**[0045]** With supporting data structures created, the method **300** may then proceed to read clusters from the cluster-based backup image and write them to the partition to be restored. Therefore, method **300** may include, at **380**, making a determination whether there is another cluster to be read from the cluster-based backup image. If the determination at **380** is No, then processing may conclude. But if the determination is Yes, then at **382**, a cluster may be read as a cluster from the cluster-based backup image and, at **384**, be written to the partition

as a cluster. As described above, data about a partition may be stored. Therefore, the restoration method **300** may include, at **390**, updating the volume bitmap to include the written cluster in a set of active clusters in the partition.

**[0046]** In one example, the method **300** is performed in the WinPE environment. Therefore, actions like reading the cluster-based backup image header data structure, writing the partition parameters to the boot record, writing the file system parameters to the boot record, resetting the bits in the volume bitmap, reading a cluster, and writing a cluster may be performed using a WinPE process. The WinPE process may be invoked through a WinPE API entry point like DeviceIoControl(..., FSCTL_GET_VOLUME_BITMAP, ...). In one example, the method **300** may be employed to restore partitions that store various file systems. For example, the partition to restore may include an NTFS file system, a FAT32 file system, and the like.

**[0047]** While **Figure 3** illustrates various actions occurring in serial, it is to be appreciated that various actions illustrated in **Figure 3** could occur substantially in parallel. By way of illustration, a first process could identify a partition to restore, a second process could locate backup images and make ready to restore from the backup images, a third process could establish and/or populate data structures associated with the partition to restore, a fourth process could read clusters, a fifth process could write clusters, and a sixth process could update the volume bitmap based on the writing. With substantially parallel processing occurring, cache mechanisms could be employed to mitigate the effects of seek and latency times. While six processes are described, it is to be appreciated that a greater and/or lesser number of processes could be employed and that lightweight processes, regular processes, threads, and other approaches could be employed.

**[0048]** **Figure 4** illustrates an example computerized method **400** for restoring a partition on a hard disk drive from a cluster-based backup image stored on the hard disk drive. The method **400** may include, at **410**, clearing a volume bitmap associated with the partition to be restored. The volume bitmap may be maintained by an operating system (e.g., Windows XP) to track which clusters in a partition on a hard disk drive are in use. Since the cluster based restoration method **400** will be writing clusters to the partition to restore, the action at **410** can facilitate starting with a logically clear partition to which clusters from the cluster-based backup image can be written. Thus, the method **400** may also include, at **420**, clearing a

partition metadata data structure (e.g., boot record) associated with the partition to be restored.

**[0049]** With the volume bitmap reset and partition metadata data structure(s) prepared, the method **400** may proceed, at **430**, to read a partition metadata parameter value(s) from the cluster-based backup image and, at **440**, to write the partition metadata parameter value(s) to the partition metadata data structure. For example, partition metadata like a partition name, size, creation date, creation time, creation entity, cluster size, file system(s) supported, and so on may be read from the cluster-based backup image and stored in the partition metadata data structure. While seven parameters are described, it is to be appreciated that a greater and/or lesser number of parameters of various types can be employed. For example, since a partition may include various file systems like an NTFS file system or a FAT32 file system, a parameter describing a file system type may be included.

**[0050]** A cluster-based backup image may include more than one cluster. Indeed it is likely to contain several thousand clusters. Thus, at **450**, a determination is made concerning whether there is another cluster to read from the cluster-based backup image. If the determination is No, then processing can conclude. But if the determination is Yes, then the method **400** can include, at **460**, reading a cluster from the cluster-based backup image, and at **470**, writing the cluster to the partition to be restored. Since an operating system (e.g., Windows XP) may store information about a partition, the method **400** may include, at **480**, updating the volume bitmap to include the cluster written to the partition to be restored as part of the restored partition. Thus, in addition to reading the image data in from the cluster-based backup image as clusters, the method **400** may also update values in various data structures (e.g., volume bitmap, boot record) while restoring a partition. The clusters may be read and/or written, and the various data structures created and/or manipulated using, for example, functionality provided by the WinPE environment.

**[0051]** In one example, the method **400** may restore the partition to be restored from a cluster-based backup image located in the partition to be restored. In another example, the method **400** may restore the partition to be restored from a cluster-based backup image stored in a different partition on the same hard disk drive.

**[0052]** While **Figure 4** illustrates various actions occurring in serial, it is to be appreciated that various actions illustrated in **Figure 4** could occur substantially in parallel.

By way of illustration, a first process could prepare operating system and/or file system data structures like a boot record and/or volume bitmap. Additionally, a second process could read clusters from a cluster-based backup image, while a third process could write the clusters to the partition to be restored and report them to the operating and/or file system via the data structures. While three processes are described, it is to be appreciated that a greater and/or lesser number of processes could be employed and that lightweight processes, regular processes, threads, and other approaches could be employed.

**[0053]** In one example, methodologies are implemented as processor executable instructions and/or operations stored on a computer-readable medium. Thus, in one example, a computer-readable medium may store processor executable instructions operable to perform a method that includes selecting a source partition on a hard disk drive, for which a cluster-based backup image of the source partition will be produced and stored in a target partition on the hard disk drive. The method may also include obtaining a boot record for the source partition using a WinPE process and examining the boot record using a WinPE process to determine a value for parameters like, a file system type, a file system size, and a cluster size associated with a file system stored in the source partition. The method may also include calculating values for source partition metadata parameters from source partition parameters, the file system type, the file system size, the cluster size, and so on. The method may also include writing a cluster-based backup image header data structure to the target partition using a WinPE process, where the cluster-based backup image header data structure includes parameters like, the source partition metadata parameters, the source partition parameters, the file system type, the file system size, the cluster size, and so on. The method may also include selectively manipulating, using a WinPE process, a bit(s) in a volume bitmap associated with the source partition, where the bit(s) is associated with a file(s) that is not to be included in the cluster-based backup image and then identifying, by accessing the volume bitmap, clusters in the source partition to be included in the cluster-based backup image. With the clusters identified, the method may also include, for the identified clusters, reading a cluster, as a cluster, from the source partition, using a WinPE process and writing the cluster, as a cluster, to the cluster-based backup image, using a WinPE process. While the above method is described being stored on a computer-readable medium, it is to be appreciated that other example methods described and claimed herein can also be stored on a computer-readable medium.

[0054] **Figure 5** illustrates an example system **500** for performing cluster-based backups and/or restores. The system **500** includes a metadata logic **510** and a data logic **520** that facilitate interacting with a hard disk drive **530** and that facilitate creating and/or restoring from a cluster-based backup image **570**. The system **500** may be temporarily, permanently, and/or removably operably connected to the disk drive **530**.

[0055] The metadata logic **510** may be configured to acquire disk drive metadata **540** that describes, for example, a hard disk drive partition **560** to be backed up. The metadata **540** may include, for example, cluster information (e.g., size, number), file system information (e.g., type, number, size), partition data (e.g., creation date, size, location), operating system information, and so on. The metadata logic **510** can be configured to write the metadata **540** and/or selected copies of portions thereof to the cluster-based backup image **570** from which the hard disk drive partition can be restored. Additionally, and/or alternatively, the metadata logic **510** can be configured to read metadata from the cluster-based backup image **570** and update the metadata **540** on the hard disk drive **530** during, for example, a restore operation.

[0056] The data logic **520** may be configured to access a volume bitmap **550** associated with the hard disk drive partition **560** to be backed up. The volume bitmap **550** may be maintained by an operating system like Windows XP. The volume bitmap **550** can be accessed and information from the volume bitmap **550** may be stored in a portion of the cluster-based backup image **530**. The data logic **520** can also be configured to selectively read clusters identified in the volume bitmap **550** as being allocated to the hard disk drive partition to be backed up **560** and to write the clusters to the cluster-based backup image **570**. The data logic **520** may also be configured to read clusters from the cluster-based backup image **570** during, for example, a restore operation.

[0057] In one example, the data logic **520** writes the cluster-based backup image **570** in the same partition on the disk drive **530** as the partition **560** to be backed up. In another example, the data logic **520** writes the cluster-based backup image **570** in a partition separate from the partition **560** to back up. Thus, the system **500** may produce the cluster-based backup image **570** to backup the partition **560** and/or may restore a partition to restore (not illustrated) from the cluster-based backup image **570**. While creating a cluster-based backup image **570** and/or restoring from the cluster-based backup image **570**, the system **500** may read and/or write to data structures that store metadata **540** (e.g., a boot record), cluster usage information (e.g., volume bitmap **550**) and so on.

[0058] Thus, the system **500**, with metadata logic **510** and data logic **520** implemented as ASICs, programs, applications, specifically programmed microprocessors, and so on, may serve as means for identifying a partition on a hard disk drive for which a cluster-based backup image is to be written onto the hard disk drive, means for acquiring and writing to the cluster-based backup image a partition metadata that describes the partition on the hard disk drive, and means for acquiring and writing to the cluster-based backup image a partition data that has a cluster(s) in the partition on the hard disk drive. Similarly, the system **500**, with metadata logic **510** and data logic **520** implemented as ASICs, programs, applications, specifically programmed microprocessors, and so on, may serve as means for identifying a cluster-based backup image on a hard disk drive from which a partition on the hard disk drive can be restored, means for acquiring and writing a cluster-based backup image partition metadata to the partition to be restored, and means for acquiring and writing to the partition to be restored a partition data that includes a cluster(s) stored in cluster-based backup image.

[0059] In one example, data structures may be constructed that facilitate storing data on a computer-readable medium and/or in a data store. Thus, in one example, a computer-readable medium may store a data structure that includes a first field containing data that forms and/or represents a cluster-based image from which a partition on a hard disk drive can be restored. The data structure may also include, for example, a second field that that stores metadata that describes the cluster-based image stored in the first field. Thus, the data structure may store the data that forms the image, and data about the image. While two fields are described, it is to be appreciated that a greater and/or lesser number of fields could be employed.

[0060] **Figure 6** illustrates a computer **600** that includes a processor **602**, a memory **604**, and input/output ports **610** operably connected by a bus **608**. In one example, the computer **600** may include a cluster-based backup and restore logic **630** configured to facilitate producing a cluster-based backup image and/or restoring a partition from the cluster-based backup image. For example, disk **606** may include a partition that a user wants to back up. The cluster-based backup and restore logic **630** can control performing cluster based reads and writes to back up the partition on disk **606** in a cluster-based backup image on disk **606**.

[0061] The processor **602** can be a variety of various processors including dual microprocessor and other multi-processor architectures. The memory **604** can include volatile memory and/or non-volatile memory. The non-volatile memory can include, but is

not limited to, ROM, PROM, EPROM, EEPROM, and the like. Volatile memory can include, for example, RAM, synchronous RAM (SRAM), dynamic RAM (DRAM), synchronous DRAM (SDRAM), double data rate SDRAM (DDR SDRAM), and direct RAM bus RAM (DRRAM).

**[0062]** A disk **606** may be operably connected to the computer **600** via, for example, an input/output interface (e.g., card, device) **618** and an input/output port **610**. The disk **606** can include, but is not limited to, devices like a magnetic disk drive, a solid state disk drive, a floppy disk drive, a tape drive, a Zip drive, a flash memory card, and/or a memory stick. Furthermore, the disk **606** can include optical drives like a CD-ROM, a CD recordable drive (CD-R drive), a CD rewriteable drive (CD-RW drive), and/or a digital video ROM drive (DVD ROM). The memory **604** can store processes **614** and/or data **616**, for example. The disk **606** and/or memory **604** can store an operating system that controls and allocates resources of the computer **600**.

**[0063]** The bus **608** can be a single internal bus interconnect architecture and/or other bus or mesh architectures. The bus **608** can be of a variety of types including, but not limited to, a memory bus or memory controller, a peripheral bus or external bus, a crossbar switch, and/or a local bus. The local bus can be of varieties including, but not limited to, an industrial standard architecture (ISA) bus, a microchannel architecture (MSA) bus, an extended ISA (EISA) bus, a peripheral component interconnect (PCI) bus, a universal serial (USB) bus, and a small computer systems interface (SCSI) bus.

**[0064]** The computer **600** may interact with input/output devices via i/o interfaces **618** and input/output ports **610**. Input/output devices can include, but are not limited to, a keyboard, a microphone, a pointing and selection device, cameras, video cards, displays, disk **606**, network devices **620**, and the like. The input/output ports **610** can include but are not limited to, serial ports, parallel ports, and USB ports.

**[0065]** The computer **600** can operate in a network environment and thus may be connected to network devices **620** via the i/o devices **618**, and/or the i/o ports **610**. Through the network devices **620**, the computer **600** may interact with a network. Through the network, the computer **600** may be logically connected to remote computers. The networks with which the computer **600** may interact include, but are not limited to, a local area network (LAN), a wide area network (WAN), and other networks. The network devices **620** can connect to LAN technologies including, but not limited to, fiber distributed data interface (FDDI), copper distributed data interface (CDDI), Ethernet (IEEE 802.3), token ring (IEEE

802.5), wireless computer communication (IEEE 802.11), Bluetooth (IEEE 802.15.1), and the like. Similarly, the network devices **620** can connect to WAN technologies including, but not limited to, point to point links, circuit switching networks like integrated services digital networks (ISDN), packet switching networks, and digital subscriber lines (DSL).

**[0066]** **Figure 7** illustrates an example printer **700** that includes a cluster-based backup and restore logic **710**. The logic **710** may be configured to perform example executable methods like those described herein. The logic **710** may be permanently and/or removably attached to the printer **700**.

**[0067]** The printer **700** may receive print data to be rendered. Thus, printer **700** may also include a memory **720** configured to store print data or to be used more generally for print image processing. Similarly, the printer **700** may include a disk drive (not illustrated). Thus, the cluster based backup and restore logic **710** may control producing a cluster-based backup image of a partition on the disk drive and/or restoring a partition on the disk drive from a cluster-based backup image.

**[0068]** The printer **700** may also include a rendering logic **730** configured to generate a printer-ready image from print data. Rendering varies based on the format of the data involved and the type of imaging device. In general, the rendering logic **730** converts high-level data into a graphical image for display or printing (e.g., the print-ready image). For example, one form is ray-tracing that takes a mathematical model of a three-dimensional object or scene and converts it into a bitmap image. Another example is the process of converting HTML into an image for display/printing. It is to be appreciated that the printer **700** may receive printer-ready data that does not need to be rendered and thus the rendering logic **730** may not appear in some printers.

**[0069]** The printer **700** may also include a print image forming mechanism **740** configured to generate a print image onto print media from the print-ready image. The print image forming mechanism **740** may vary based on the type of the printer **700** and may include a laser imaging mechanism, other toner-based imaging mechanisms, an ink jet mechanism, digital imaging mechanism, or other imaging reproduction engine. A processor **750** may be included that is implemented with logic to control the operation of the printer **700**. In one example, the processor **750** includes logic that is capable of executing Java instructions. Other components of the printer **700** are not described herein but may include

media handling and storage mechanisms, sensors, controllers, and other components involved in the imaging process.

[0070] Referring now to **Figure 8**, an application programming interface (API) **800** is illustrated providing access to a system **810** for producing a cluster-based backup image and/or restoring a partition on a hard disk drive from a cluster-based backup image. The API **800** can be employed, for example, by programmers **820** and/or processes **830** to gain access to processing performed by the system **810**. For example, a programmer **820** can write a program to access the system **810** (e.g., invoke its operation, monitor its operation, control its operation) where writing the program is facilitated by the presence of the API **800**. Rather than programmer **820** having to understand the internals of the system **810**, the programmer **820** merely has to learn the interface to the system **810**. This facilitates encapsulating the functionality of the system **810** while exposing that functionality.

[0071] Similarly, the API **800** can be employed to provide data values to the system **810** and/or retrieve data values from the system **810**. For example, a process **830** that processes cluster-based backup image metadata can provide the metadata to the system **810** via the API **800** by, for example, using a call provided in the API **800**. Thus, in one example of the API **800**, a set of application programming interfaces can be stored on a computer-readable medium. The interfaces can be employed by a programmer, computer component, logic, and so on to gain access to a system **810** for producing a cluster-based backup image and/or restoring a partition on a disk drive from a cluster-based backup image. The interfaces can include, but are not limited to, a first interface **840** that communicates a data concerning a partition to restore, and a second interface **850** that communicates a data concerning a cluster-based backup image.

[0072] Thus, in one example, a set of application programming interfaces may be embodied on a computer-readable medium for execution by a computer component in conjunction with producing a cluster-based image backup and/or restoring a partition from a cluster-based image. The set of APIs may include, for example, a first interface for identifying a partition on a hard disk drive for which a cluster-based backup image is to be written onto the hard disk drive and a target partition on the hard disk drive in which the cluster-based backup image is to be placed. The set of APIs may also include, for example, a second interface for identifying a cluster-based backup image on the hard disk drive from

which a partition on the hard disk drive can be restored and a target partition on the hard disk drive to be restored.

**[0073]** While example systems, methods, and so on have been illustrated by describing examples, and while the examples have been described in considerable detail, it is not the intention of the applicants to restrict or in any way limit the scope of the appended claims to such detail. It is, of course, not possible to describe every conceivable combination of components or methodologies for purposes of describing the systems, methods, and so on described herein. Additional advantages and modifications will readily appear to those skilled in the art. Therefore, the invention, in its broader aspects, is not limited to the specific details, the representative apparatus, and illustrative examples shown and described. Accordingly, departures may be made from such details without departing from the spirit or scope of the applicants' general inventive concept. Thus, this application is intended to embrace alterations, modifications, and variations that fall within the scope of the appended claims. Furthermore, the preceding description is not meant to limit the scope of the invention. Rather, the scope of the invention is to be determined by the appended claims and their equivalents.

**[0074]** To the extent that the term "includes" or "including" is employed in the detailed description or the claims, it is intended to be inclusive in a manner similar to the term "comprising" as that term is interpreted when employed as a transitional word in a claim. Furthermore, to the extent that the term "or" is employed in the detailed description or claims (e.g., A or B) it is intended to mean "A or B or both". When the applicants intend to indicate "only A or B but not both" then the term "only A or B but not both" will be employed. Thus, use of the term "or" herein is the inclusive, and not the exclusive use. See, Bryan A. Garner, A Dictionary of Modern Legal Usage 624 (2d. Ed. 1995).